Assignment: Build a Customer Support Bot with Document Training

Objective

Design and implement an agentic workflow in Python that trains on a provided document (e.g., a PDF or text file), answer customer support queries based on that document, and refines its responses using simulated feedback (e.g., "not helpful," "too vague"). The bot should demonstrate autonomy, decision-making, and iterative improvement.

Problem Statement

You are tasked with creating a customer support bot that:

1. Reads and processes a provided document (e.g., a company FAQ or product manual).
2. Uses a pre-trained NLP model to generate responses to customer queries based on the document.
3. Evaluates its responses using simulated feedback and adjust its strategy (e.g., adding more detail or rephrasing).
4. Logs its actions and decisions for transparency.
5. Handles cases where the query isn't covered by the document gracefully.

Requirements

- **Language**: Python
- **Expected Technologies**:
  - **Document Processing**: PyPDF2 (for PDFs) or plain text reading with Python's open() function
  - **NLP Model**: Hugging Face transformers library (e.g., distilbert-base-uncased for question-answering or a text generation model like gpt2)
  - **Text Embedding/Search**: sentence-transformers (to find relevant sections of the document) or simple keyword matching
  - **Workflow Management**: Python classes for agent logic (optional: LangChain for advanced agentic behavior)
  - **Logging**: logging module to track decisions and actions
  - **Feedback Simulation**: Custom rules or random feedback generator (e.g., "not helpful")
- **Input**: A sample document (e.g., a 1-2 page FAQ in PDF or TXT format) provided by the user or created by you (e.g., a fake company FAQ).
- **Output**:
  - A Python script or notebook that runs the bot.
  - A log file (support_bot_log.txt) showing the bot's decisions and iterations.
  - Sample query responses printed to the console.

Tasks

1. **Document Training**:
   - Load and preprocess the provided document (e.g., extract text from a PDF or TXT file).
   - Split the document into sections or sentences for easier retrieval (e.g., by paragraphs or Q&A pairs).
   - Optionally, create embeddings of the document sections using sentence-transformers for semantic search.
2. **Query Handling**:
   - Accept a customer query as input (e.g., "How do I reset my password?").
   - Use a pre-trained NLP model (e.g., question-answering or text generation) to find and generate a response based on the document.
   - If embeddings are used, retrieve the most relevant section; otherwise, use keyword matching.
3. **Feedback Loop**:
   - Simulate feedback with a function (e.g., randomly select from: "not helpful," "too vague," "good").
   - Adjust the response strategy based on feedback (e.g., provide more context for "too vague" or rephrase for "not helpful").
   - Limit to 2 iterations per query.
4. **Agent Logic**:
   - Create a Python class SupportBotAgent to manage the workflow.
   - Include methods for training on the document, answering queries, evaluating responses, and adjusting.
   - Log each step (e.g., "Loaded document: FAQ.pdf", "Query: [query]", "Feedback: too vague").
5. **Execution**:
   - Test the bot with 3-5 sample queries (e.g., "How do I contact support?", "What's the refund policy?").
   - Handle queries not covered by the document with a fallback (e.g., "I don't have enough information to answer that").

Deliverables

- A Python script (e.g., support_bot_agent.py) or Jupyter Notebook.
- A log file (support_bot_log.txt) with the bot's decisions and actions.
- A short README (text file) explaining how to run the code, provide a document, and any setup steps (e.g., installing dependencies).

Submission Guidelines

1. Provide a GitHub repository with:

   - Clear README with setup instructions
   - Development decisions documentation
   - Any known issues or future improvements

2. Deploy the application
3. Time limit: 1 week

Sample Starter Code

Here’s a skeleton to help get started (assumes a text file input and uses a question-answering model):

python
CollapseWrapCopy
```
import logging
from transformers import pipeline
from sentence_transformers import SentenceTransformer, util
import random

# Set up logging
logging.basicConfig(filename='support_bot_log.txt', level=logging.INFO)

class SupportBotAgent:
    def __init__(self, document_path):
        self.qa_model = pipeline("question-answering", 
model="distilbert-base-uncased-distilled-squad")
        self.embedder = SentenceTransformer('all-MiniLM-L6-v2')
        self.document_text = self.load_document(document_path)
        self.sections = self.document_text.split('\n\n')  # Split by 
paragraphs
        self.section_embeddings = self.embedder.encode(self.sections, 
convert_to_tensor=True)
        logging.info(f"Loaded document: {document_path}")

    def load_document(self, path):
        with open(path, 'r', encoding='utf-8') as file:
            text = file.read()
        return text

    def find_relevant_section(self, query):
        query_embedding = self.embedder.encode(query, 
convert_to_tensor=True)
        similarities = util.cos_sim(query_embedding, 
self.section_embeddings)[0]
        best_idx = similarities.argmax()
        logging.info(f"Found relevant section for query: {query}")
        return self.sections[best_idx]
    def answer_query(self, query):
        context = self.find_relevant_section(query)
        if not context:
            return "I don’t have enough information to answer that."
        result = self.qa_model(question=query, context=context)
        return result["answer"]

    def get_feedback(self, response):
        feedback = random.choice(["not helpful", "too vague", "good"])
        logging.info(f"Feedback received: {feedback}")
        return feedback

    def adjust_response(self, query, response, feedback):
        if feedback == "too vague":
            context = self.find_relevant_section(query)
            return f"{response} (More info: {context[:100]}...)"  # Add
context
        elif feedback == "not helpful":
            return self.answer_query(query + " Please provide more
details.")  # Rephrase
        return response

    def run(self, queries):
        for query in queries:
            logging.info(f"Processing query: {query}")
            response = self.answer_query(query)
            print(f"Initial Response to '{query}': {response}")

            # Feedback loop (max 2 iterations)
            for _ in range(2):
                feedback = self.get_feedback(response)
                if feedback == "good":
                    break
                response = self.adjust_response(query, response, feedback)
                print(f"Updated Response to '{query}': {response}")

if __name__ == "__main__":
    # Sample document should be provided as 'faq.txt'
    bot = SupportBotAgent("faq.txt")
    sample_queries = [
        "How do I reset my password?",
        "What’s the refund policy?",
        "How do I fly to the moon?"  # Out-of-scope query
    ]
    bot.run(sample_queries)
```

Sample FAQ Document (faq.txt)

text

CollapseWrapCopy

```
Resetting Your Password
To reset your password, go to the login page and click "Forgot Password."
Enter your email and follow the link sent to you.

Refund Policy
We offer refunds within 30 days of purchase. Contact support at
support@example.com with your order number to start the process.

Contacting Support
Email us at support@example.com or call 1-800-555-1234 during business
hours (9 AM - 5 PM EST).
```

Evaluation Criteria

- **Functionality**: Does the bot train on the document and answer queries accurately?
- **Adaptability**: Does it adjust responses based on feedback?
- **Code Quality**: Is the code modular, readable, and well-commented?
- **Logging**: Are key steps and decisions logged appropriately?
- **Robustness**: Can it handle out-of-scope queries gracefully?
- **Documentation**: Is the README clear and sufficient?

Notes for the Engineer

- Install dependencies: pip install transformers sentence-transformers PyPDF2 (add langchain if used).
- Provide a sample document (faq.txt) or adapt the code for PDF input with PyPDF2.
- Experiment with different NLP models or retrieval methods (e.g., keyword search vs. embeddings).